WSysDynEng.cls	System dynamics Engine class
wSysDynEng.bas	System dynamics Engine module (this module was introduced only for speed purposes because all the code should theoretically be encapsulated in the class)
wConst.bas	Intelligent Coaching Agent constant declaration
wDeclare.bas	Intelligent Coaching Agent DLL interface
wIca.cls	Intelligent Coaching Agent class
wIca.bas	Intelligent Coaching Agent module (this module was introduced only for speed purposes because all of the code should theoretically be encapsulated in the class)

To utilize the system dynamics engine fully, the developer must place code in different strategic areas or stages of the application. Initial stage - the loading of the form containing the simulation front-end. This is when the simulation model and system dynamic engine are initialized. Modification stage - Takes place when the user makes changes to the front-end that impacts the simulation model PInputs). This is when the ICA is notified of what's happening. Run stage - The system dynamics model is run and parameter outputs are received. Feedback stage - The user requests feedback on the work that they have performed. This is when the simulation notifies the ICA of all output changes. Final stage - The simulation front-end unloads. This is when the simulation model is saved. These stages will be explained by including the Visual Basic code involved as well as a short description of that code.

Initial Stage Code In Accordance With A Preferred Embodiment

1. Creating the ICA & the simulation engine objects: Code: Set moSysDynEngine = New classSysDynEngine; Set moICA = New classICA; Description: The first step in using the system dynamics engine is to create an instance of the classSysDynEngine class and also an instance of the classICA class. Note that the engine and ICA should be module level object "mo" variables. **2. Loading the simulation**: Code: IRet = moSysDynEngine.OpenSimulation(FILE_SIM, Me.bookSim, True); IRet = moSysDynEngine.LoadSysDyn(mIICATaskID, DB_SIMULATION, 1); IRet = moSysDynEngine LoadModel(MODEL_NAME,mbTaskStarted); Description: After the object creation, the OpenSimulation, LoadSimulation and LoadModel methods of the system dynamics engine object must be called. The OpenSimulation method reads the specified Excel 5.0 spreadsheet file (FILE_SIM) into a spreadsheet control (bookSim). The LoadSysDyn method opens the simulation database (DB_SIMULATION) and loads into memory a list of parameter inputs and a list of parameter outputs. The LoadModel method opens a system dynamics model (MODEL_NAME). Every method of the system dynamics engine will return 0 if it completes successfully otherwise an appropriate error number is returned. **3. Initializing and loading the Intelligent Coaching Agent**: Code: IRet = moICA.Initialize(App.Path & "\" & App.EXEName & ".ini", App.Path & DIR_DATABASE, App.Path & DIR_ICADOC, App.Path & "\"); IRet = moICA.LoadTask(mIICATaskID, ICAStudentStartNew); Description: The system dynamics engine only works in conjunction with the ICA. The Initialize method of the ICA object reads the application .ini file and sets the Tutor32.dll appropriately. The LoadTask method tells the ICA (Tutor32.dll) to load the .tut document associated to a specific task in memory. From that point on, the ICA can receive notifications. Note: The .tut document contains all the element and feedback structure of a task. Ex: SourcePages, SourceItems, TargetPages, Targets, etc... **4. Restoring the simulation**- Code: IRet = moSysDynEngine.RunPInputs(MODEL_NAME, True); IRet = moSysDynEngine.RunPOutputs(MODEL_NAME, True); IRet = moSysDynEngine.PassPInputsAll; Call moICA.Submit(0); Call

mol CA.SetDirtyFlag(0, False) Description: Restoring the simulation involves many things: clearing all of the parameter inputs and outputs when the user is starting over; loading the interface with data from the simulation model; invoking the PassPInputsAll method of the system dynamics engine object in order to bring the ICA to its original state; invoking the RunPinputs and RunPOutputs methods of the system dynamics engine object in order to bring the system dynamics model to it's original state; calling the Submit method of the ICA object to trigger the ICA to play all of the rules; calling the SetDirtyFlag of the ICA object to reset the user's session. Running parameters involves going through the list of TutorAware PInputs and POutputs and notifying the ICA of the SourceItemID, TargetID and Attribute value of every one.Modification Stage; **1. Reading parameter inputs & outputs;** Code: Dim sDataArray(2) as string; Dim vAttribute as variant; Dim lSourceItemID as long, lTargetID as long; lRet = moSysDynEngine.ReadReference("Input_Name", vAttribute, lSourceItemID, lTargetID, sDataArray). Description: The ReadReference method of the system dynamics object will return the attribute value of the parameter input or output referenced by name and optionally retrieve the SourceItemID, TargetID and related data. In the current example, the attribute value, the SourceItemID, the TargetID and 3 data cells will be retrieved for the parameter input named Input_Name.

2. Modifying parameter inputs Code: Dim vAttribute as variant; Dim lSourceItemID as long; Dim sDataArray(2) as string; vAttribute=9999; sDataArray(0)="Data Cell #1"; sDataArray(1)="Data Cell #2"; sDataArray(2)="Data Cell #3"; lRet = moSysDynEngine.WriteReference("Input_Name", vAttribute, , sDataArray). Description: To modify a parameter input, call the WriteReference method of the system dynamics object and pass the PInput reference name, the new attribute value and optionally a data array (an additional information to store in the simulation model). The system dynamics engine notifies the ICA of the change. Run Stage **1. Playing the System Dynamics Model;** Code: lRet = moSysDynEngine.PlayModel(SYSDYN_PLAYSTEP); lblCurrentTime.Caption = moSysDynEngine.CurrentTime; and lblLastTime.Caption = moSysDynEngine.LastTime; Description: Playing the system dynamics model is also handled by the system dynamics engine. There are three ways that the models can be played, all at once, one step at a time (shown above) or until a specific point in time. These are the parameters that are passed into the PlayModel method. Playing of the model generates the parameter output values and passes the Tutor Aware POutputs to the ICAT. The engine also keeps track of time and these values can be read using the CurrentTime and LastTime properties. **2. Jumping Back in a System Dynamics Model** Code: lRet = mol CA.LoadTask(mlICATaskID, ICAStudentStartNew); lRet = moSysDynEngine.JumpBack(TIME_TO_JUMP_TO). Description: Because the system dynamics engine writes backup copies of the parameters passed to and from it, it can start over and resubmit these values back to the system dynamics model until a given period of time. To do this, the code would need to restart the ICA and then call the system dynamics engine to jump back to a given time (TIME_TO_JUMP_TO). Feedback stage **1. Triggering the ICA Rule engine;** Code: lRet= mol CA.Submit(lCoachID); Description: Once the simulation has been processed, the Submit method of the ICA object must be called to trigger all the rules and deliver the feedback. This feedback will be written by the Tutor32.dll to two RTF formatted files. One file for previous feedback and one file for the current feedback.

ICA Configuration in Accordance with a Preferred Embodiment

Figure **28** is an overview diagram of the logic utilized for initial configuration in accordance with a preferred embodiment. Since the structure of the feedback is the same as other on-line activities, the ICA can also be configured in the same manner. For ease of creation and maintenance of ICA feedback, it is recommended that the feedback is constructed so that only one rule fires at any point in time. Note that the organization of the example is one of many ways to structure the feedback. Step 1: Create a map of questions and follow-up questions; Before designers start configuring the ICA, they should draw a map of the questions, videos and follow-up questions that they wish to use in the on-line meeting. This will give them a

good understanding of the interactions as they configure the ICA. Step 2: Create a coach; All feedback is given by a coach. Create a specific coach for the on-line meeting. Step 3: Create the Source Items and Targets

Every question will have one Source Item (1) and Target (2) associated with it. These will be used by the ICA to show videos and follow-up questions. For organizational purposes and ease of reading, it is recommended that each Source Page ("0 Intro") contain all of the follow up questions ("Intro Q1", "Intro Q2", "Intro Q3"). Targets can be created one per Source Item (shown here) or one per many Source Items. This is not very important, so long as there are distinct Source Item and Target associations. Once the Source Items and Targets have been created, associate them into SourceItemTargets (3) and give them a relevance of one. These are the unique identifiers which the ICA will use to fire rules and to provide feedback to the student. Step 4: Create the Parent Header (Video Information) Figure **29** is a display of video information in accordance with a preferred embodiment. Feedback (Coach Items) are organized into Target Groups (1). In Figure **29**, each on-line question has one Target Group for ease of maintenance. Each TargetGroup must have at least one related Target (4). These are the SourceItemTarget mappings that were made at the end of Step 3. Next, Rules (2) are created to fire when the SourceItemTarget is mapped (a question is clicked). Coach Items (3) are associated to a rule and represent the feedback which will be shown if the rule is fired. The ICA Utilities incorporate business simulation into a multimedia application. What this means is that there is now a middle layer between the application and the ICAT. These utilities, along with the simulation engine (described later), allow the architecture to be a front end to the simulation. Now, any changes to a simulation model do not need to be incorporated into code. The ICA Utilities and simulation engine work with simulation models created in Microsoft Excel. After the model is created, the designer uses the Defined Name function in Excel to flag specific cells that are to be used by the application and the ICA Utilities in accordance with a preferred embodiment. Figure **30** illustrates an ICA utility in accordance with a preferred embodiment. The ICA Utilities consist of six utilities that work with the Intelligent Coaching Agent Tool (ICAT) to incorporate business simulation with the multimedia application.